Letters

Fuzzy Rule-Based Networks for Control

Charles M. Higgins and Rodney M. Goodman

Abstract—**We present a method for learning fuzzy logic membership functions and rules to approximate a numerical function from a set of examples of the function's independent variables and the resulting function value. This method uses a three-step approach to building a complete function approximation system: first, learning the membership functions and creating a cell-based rule representation; second, simplifying the cell-based rules using an information-theoretic approach for induction of rules from discrete-valued data; and, finally, constructing a computational (neural) network to compute the function value given its independent variables. This function approximation system is demonstrated with a simple control example: learning the truck and trailer backer-upper control system.**

I. INTRODUCTION

THE problem of approximating a function from a set of examples can be solved in a multitude of ways, including mathematical methods using an explicit model for the function to be learned and model-free systems such as neural networks and fuzzy systems. The flexibility and wide applicability of model-free systems has led to wide interest in their use, particularly in learning control system functions. The ability of fuzzy systems to express complex functions in terms of linguistic rules makes such systems an attractive alternative to neural network "black boxes," in which the function learned can only be observed through the input/output relationship.

While there are well-known methods in existence for the approximation of functions using neural networks (two of the most successful are backpropagation [1] and radial basis functions [2]), methods for creating fuzzy systems from data are less well developed. The approach of Kosko [3] learns only the rules, requiring the membership functions to be set up by hand. Lin [4] and, later, Horikawa [5] and d'Alché Buc [6] start with a fixed number of rules and membership functions and perturb them by backpropagation until they fit the data. An *a priori* choice of the number of rules and membership functions is required by these approaches. Similar to the approach of the radial basis function network, Wang [7] learns to express a function in terms of fuzzy basis functions. Since each basis function has its own set of membership functions, the explanation ability of the rule-based system is mostly forfeited. Sugeno and Kang [8] present a method for "fuzzy structure identification" which constructs (evenly spaced) membership functions and rules with a functional conclusion. While this method is a powerful function approximator, again the explanation ability of this system is limited.

In this paper, we present a novel method for learning a fuzzy system to approximate example data. The membership functions and a minimal set of rules are constructed automatically from the example data, and the final system is expressed as a computational (neural) network for efficient parallel computation of the function value. This method does not require the convergence of an iterative energy search algorithm, as in backpropagation methods, and retains the explanation ability of rule-based systems by expressing the example data in terms of simple rules and a single set of membership functions.

Manuscript received December 2, 1992; revised March 8, 1993. This work was supported in part by Pacific Bell, and in part by DARPA and ONR under Grant N00014-92-J-1860.

C. M. Higgins is with MIT Lincoln Laboratory, Lexington, MA 02173.

R. M. Goodman is with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125.

IEEE Log Number 9213138.

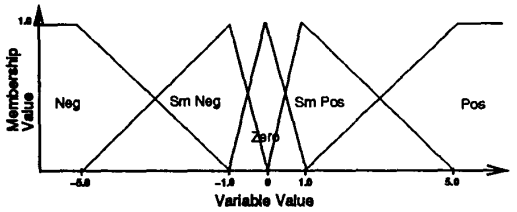


Fig. 1. Membership function example.

The proposed learning algorithm can be used to construct a fuzzy control system from examples of an existing control system's actions. This can be useful in converting any existing controller into a fuzzy controller. The learned controller shares the advantages of all fuzzy systems—it can be easily modified via the membership functions and rules if the performance is unsatisfactory, and the behavior of the controller can be explained directly in terms of fuzzy rules.

II. FUZZY LOGIC FRAMEWORK

Fuzzy logic is still a developing field. There is still much disagreement in the literature about the best way each fuzzy primitive should be realized. In this section, we describe and justify the choices we have made to define our fuzzy system.

A. Membership Functions

Within the framework of "fuzzy logic," there is considerable leeway in the choice of membership function shape and overlap. No clearly optimal choices exist; however, the following assumptions make the learning process much more well posed. We will use piecewise linear membership functions rather than Gaussian or other continuous functions; such membership functions are simple to implement and conputationally efficient. We will also specify that membership functions are *fully overlapping*; that is, at any given value of the variable, the total membership sums to one. See Fig. 1 for an example of both properties. Given these two properties of the membership functions, we need only specify the positions of the peaks of the membership functions to completely describe them. Another benefit of these choices for membership functions is that they allow the interpretation of the system as a simple interpolation between points in the input space. If all rules had a value for every input variable on their condition side, then each rule would specify the value of the output at a single point in the input space, and the system would interpolate smoothly between these points to determine the complete output surface. Depending on the number of conditions of a rule, it may also specify a line, a plane, or a hyperplane in the input space.

B. Fuzzy Rules

We define a fuzzy rule as *if* y *then* x, where y(the condition side) is a conjunction in which each clause specifies an input variable and one

of the membership functions associated with it, and x(the conclusion side) specifies an output variable membership function. There may be at most one clause for each input variable. Thus, an example rule is

$$\text{if input1} = \text{high and input2} = \text{low then output} = \text{medium}.$$

If a set of rules has a clause for *every* input variable on the condition side of each rule, we call it a *cell-based* rule set, because any combination of membership functions for every input variable defines a *cell* in the input space.

C. Fuzzy Inference

There are three fuzzy primitives needed to do inference with the membership functions and rules we have described above. The firing strength of each rule is calculated as a *Fuzzy* AND of its conditions; the weight given to each output membership function is calculated as a *Fuzzy* OR of the firing strengths of each rule which leads to that conclusion; and finally, the crisp final output is calculated as a *defuzzification* of the weights for each output membership function.

1) Fuzzy AND: We will define Fuzzy AND as a product. A product gives a smoother tradeoff between rules than using the minimum, more common in the fuzzy literature. Use of the minimum results in a sharp corner in the output where the minimum stops following one input and begins to follow the other. The smoother response of the product is better for a simple interpolative function approximation system; the lack of sharp edges is particularly good for a smooth control response. However, the more inputs there are, the less the product looks like a minimum; for a large number of inputs, the product looks like a crisp AND. By using this definition, we are implicitly assuming that the number of inputs is relatively small.

2) Fuzzy OR: We will define Fuzzy OR as a (normalized) sum. A more common approach in the fuzzy literature is to use the maximum rule weight. However, summing the weights rather than taking the maximum results in a smoother output surface. Again, this is better for a function approximation system.

3) Defuzzification: For defuzzification, we will employ the *singleton* method, proposed by Sugeno [9], which utilizes only the weights w_i for each output fuzzy set and the peaks P_i of each fuzzy set membership function. The crisp output is calculated as

$$O = \frac{\sum w_i P_i}{\sum w_i}.$$

Note that the shape of the output membership functions is not used in output computation—only the peaks; thus the output membership functions can be considered as "spikes," or *fuzzy singletons*. This method is computationally efficient and allows a simple network implementation.

III. Learning a Fuzzy System from Example Data

There are three steps in our method for constructing a fuzzy system: first, learn the membership functions and an initial rule representation; second, simplify (compress) the rules as much as possible using information theory; and, finally, construct a computational network with the rules and membership functions to calculate the function value given the independent variables.

Hereafter, we will refer to the function value as the output variable, and the independent variables of the function as the input variables.

A. Learning the Membership Functions

Before learning, two parameters must be specified. First, the maximum allowable rms error of the approximation from the example data; second, the maximum number of membership functions for each variable. The system will not exceed this number of membership functions, but may use fewer if the error is reduced sufficiently before the maximum number is reached. If the maximum allowable rms error is unknown, this parameter can be set to zero and all of the allowed membership functions will be used.

1) The Successive Approximation Algorithm: The following steps are performed to construct membership functions and a set of cell-based rules to approximate the given data set. Initially, there are no membership functions.

An example is provided in Fig. 3 of learning the function in Fig. 2.

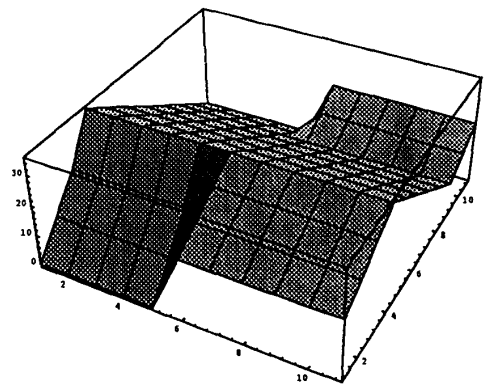


Fig. 2. Function to be learned.

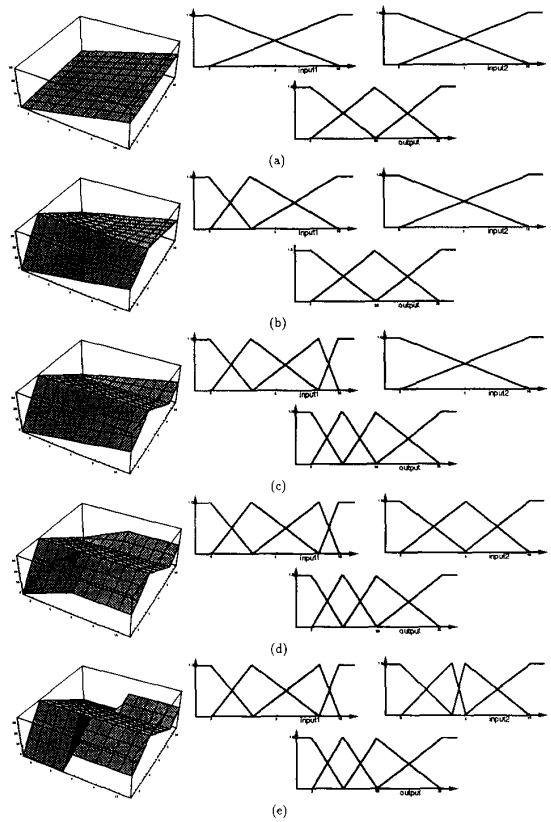


Fig. 3. Successive approximations to target function.

1) **Set up initial model.**
   a) **Add input membership functions at input extrema.** We add membership functions for each input variable at its maximum and minimum in the data set. Fig. 3(a) shows the input membership functions representing the input extrema in our example.
   b) **Add output membership functions at the corner points.** A "corner" of the input space is a point at which each of the input variables is at its maximum or minimum value in the data set. The closest example point to each corner is found and a membership function for the output is added at its value at the corner point. Fig. 3(a) shows the three membership functions obtained for the output by looking at the corners of the example function.
   c) **Create the initial rule set.** The initial cell-based rule set contains a rule for each corner, specifying the closest output membership function to the actual value at that corner. Each rule effectively represents the point that was closest to that corner. Thus we begin with a planar (hyper-planar) model of the system. Note that this is **not** the best planar approximation to the data, but merely the plane correct at the corners. Fig. 3(a) shows the initial planar approximation to the example function.
2) **Add membership functions at the point of maximum error.** We compare the current model to the function to be learned and find the example point with the maximum absolute error. We then add a membership function *for each variable* at its value at the point of maximum error. This allows us to completely specify the point, thus totally eliminating its error. (In this paper, we assume that the input data is noiseless; if there is noise, a more complex scheme for choosing this point may be necessary.) Fig. 3(b)–(e) shows the membership functions added at the point of maximum error for four iterations, gradually improving the approximation to the example function.
3) **Construct a new cell-based rule set; update output membership functions.** In this step, we construct a new set of rules to approximate the function. Constructing rules simply means determining the output membership function to associate with each cell. While constructing this rule set, we will also add any output membership functions which are lacking in the data; note that when we add a single new membership function, we add a number of rules to the cell-based set. The correct output value for any point which was not explicitly added may not be among the output membership functions. The best rule for a given cell is found by finding the closest example point to the rule (recall each rule specifies a point in the input space). If the output value at this point is "too far"[1] from the closest output membership function value, this output value is added as a new output membership. After this addition has been made, if necessary, the closest output membership function to the value at the closest point is used as the conclusion of the rule.
4) **If error threshold has been reached or all membership functions are full, exit. Otherwise, go back to step 2.** By Fig. 3(e), the rms error of the model from the example function is so small that the algorithm can terminate.

2) Control System Considerations: In a general function approximation system, we are concerned with error in all parts of the input space. However, if we are learning a control system, we are more concerned with precision in the approximation near the "zero-error" or "goal" state. It is acceptable if the approximation is less precise far away from the goal state, as long as the control system is able to get the plant near the goal state. Near the goal state, we require more precision in order to have a satisfactory result. This uneven requirement for precision is usually expressed by fuzzy control system designers by putting more membership functions near the goal state. To allow for this requirement, in finding the point with the maximum error in the algorithm given above we multiply the error calculated for each point by a weighting factor which is inversely proportional to the distance from the goal state. This will result in more membership functions near the goal state. For the experimental results shown in this paper, we used a function which decreases exponentially with distance from the goal state; the severity of this decrease is set high if the function to be learned contains much complexity irrelevant to the control problem (such as in the truck backer-upper example in Section IV) and is set low if most of the details of the example function are important. The following function has been used successfully:

$$W(D, D_{max}, M) = e^{D \log(M)/D_{max}}$$

where D is the Euclidean distance from the goal state, D_{max} is the maximum distance from the goal state, and M is the desired weight at the maximum distance.

As an additional measure to assure a precise response at the goal state, we add membership functions before learning at the goal state in each variable. This assures that the system will know exactly what to do at the goal state, rather than bouncing back and forth between points on either side.

B. Simplifying the Rules

In order to have as simple a fuzzy system as possible, we would like to use the minimum possible number of rules. The cell-based rule set resulting from the membership function learning step may contain many more rules than are necessary to represent the data. These rules can be "compressed" into a set of rules which are not cell-based—that is, they may have fewer conditions than there are input variables. This compressed rule set will approximate the same function as the original cell-based rule set.

We propose the use of an information-theoretic algorithm for induction of rules from a discrete data set [10] for this purpose. The key to the use of this method is the interpretation of each of the original cell-based rules as an example from a discrete example set. The cell-based rule set becomes a discrete data set which is input to a rule-learning algorithm. This algorithm learns the best rules to describe the data set.

There are two components of the rule-learning scheme. First, we need a way to tell which of two candidate rules is the best. Second, we need a way to search the space of all possible rules in order to find the best rules without simply checking every rule in the search space.

1) Ranking Rules: Smyth and Goodman [11] have developed an information-theoretic measure of rule value with respect to a given discrete data set. This measure is known as the j-measure; defining a rule as *if* y *then* X where y is a conjunction of input variable values and X is a value of the output variable, the j-measure can be expressed as follows:

$$j(X|y) = p(X|y)\log_2\left(\frac{p(X|y)}{p(X)}\right) + p(\overline{X}|y)\log_2\left(\frac{p(\overline{X}|y)}{p(\overline{X})}\right).$$

The probabilities are computed from relative frequencies counted in the given discrete data set. The j-measure is a pure "goodness" measure in that it values only the correctness of the rule. Reference [11] also suggests a modified rule measure, the J-measure:

$$J(X|y) = p(y)j(X|y).$$

[1] Defined as a fixed percentage of the range of the output variable.

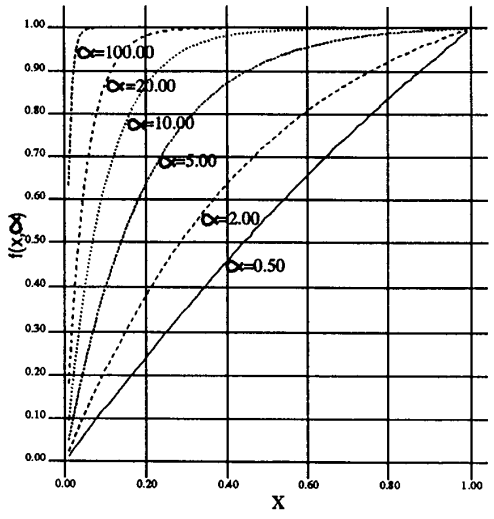


Fig. 4. Graph of $f(x, \alpha)$ for different α's.

This measure uses a multiplicative simplicity term to discount rules which are not as useful in the data set in order to remove the effects of "noise" or randomness. This has the effect of bringing out the underlying pattern in the data.

The measures shown above have been developed for discrete classifier data sets. For the application of these measures to compression, we wish to vary the rule simplicity term between that of the two measures. This allows us to get more compression than the j-measure would allow, but also ensures that we do not get so much error that our approximated function becomes significantly different. We thus propose the following rule "goodness" measure, which allows a gradual variation of the amount of noise tolerance (see Fig. 4):

$$L(X|y) = f(p(y), \alpha) j(X|y)$$

where

$$f(x, \alpha) = \frac{1 - e^{-\alpha x}}{1 - e^{-\alpha}}.$$

The parameter α may be set at ∞ to obtain the j-measure, since

$$\lim_{\alpha \to \infty} f(x, \alpha) = 1 \qquad (x > 0)$$

or at 0^+ to obtain the J-measure, since

$$\lim_{\alpha \to 0} f(x, \alpha) = x.$$

Any value of α between 0 and ∞ will result in an amount of compression between that of the J-measure and the j-measure; thus, if we are able to tolerate some error in the prediction of the original rule set, we can obtain more compression that the j-measure could give us, but not as much as the J-measure would require. Consider the example shown in Fig. 5. In this case, as we vary the parameter α in the L-measure from large (the j-measure) to small (the J-measure), the error in predicting the original rule set (treated as a discrete data set) holds at near zero for some time before increasing. By the time the error has reached 5%, more than 30% compression of the original rules has been obtained. The J-measure goes too far, causing an intolerable 15% error in prediction of the original rule set.

2) Searching for the Best Rules: Given a way to numerically rank rules, we need a way to search the space of all possible rules in such a way as to select the best rules without covering the entire space, whose size is exponential in the number of input

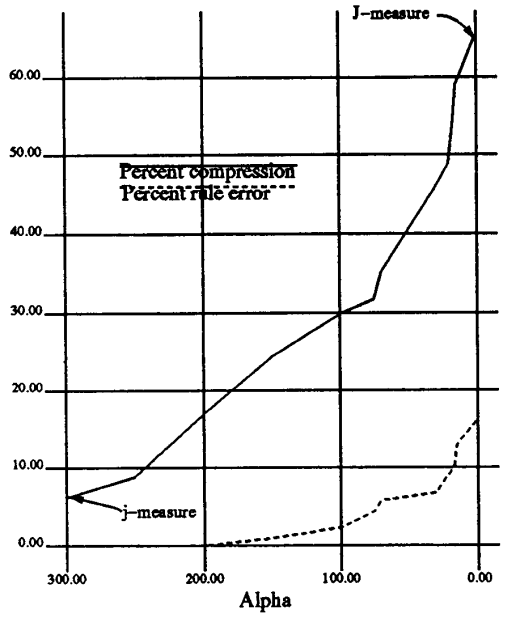


Fig. 5. Compression versus error in rule prediction.

variables. Several search algorithms have been proposed, including a constrained search of all the possible rules (ITRULE [12]). The following search algorithm [10] searches a smaller subset of the space than previous algorithms by using the examples directly as templates for rules.

Given a training set of discrete examples, an obvious way to predict the output for a novel combination of inputs is to retain all the examples and match an incoming example to an example in storage. This is equivalent to regarding the examples as very specific rules. However, these rules will not match any example not explicitly contained in the training set. Consider now if we could decide which input variables in each example to remove in order to generalize the examples to rules which cover more of the original examples; the information measures discussed above provide just such a way.

The algorithm for rule generation is as follows. Create an initial rule from each example. If there are N input variables, each initial rule is of *order* (number of conditions) N. To develop the best rule from this example, do the following.

1) Calculate the goodness measure for the rule. Call this rule the parent rule.
2) For each of the child rules generated by removing a single input variable condition from the parent rule, calculate the goodness measure (if the parent rule was order K, each of the K child rules is order $K - 1$).
3) Choose the rule among the parent rule and the set of child rules with the greatest goodness measure. Special cases:
   a) if two rules have the same goodness measure, choose the one with the lower order;
   b) if two rules of the same order have the same goodness measure, choose a random one.
4) If the chosen rule is not the parent rule, the chosen rule becomes a new parent rule; repeat the process starting at step 1. If the chosen rule is the parent rule, terminate.

C. Constructing a Network

Constructing a computational network to represent a given fuzzy

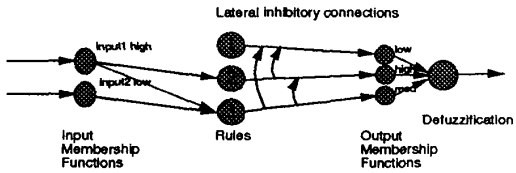


Fig. 6. Computational network constructed from fuzzy system.

system can be accomplished as shown in Fig. 6. From input to output, layers represent input membership functions, rules, output membership functions, and finally defuzzification. A novel feature of our network is the lateral links shown in Fig. 6 between the outputs of various rules. These links allow inference with dependent rules. Each layer is described in detail below.

1) The Input Membership Layer: This layer merely implements the input membership functions by generating a value between zero and one given a numerical input. A connection is made into each node in this layer from the input variable for which it represents a membership function.

2) The Rule Layer: This layer contains a node for each rule, receiving inputs from the appropriate input layer membership functions, and connecting to exactly one output membership function node. Each node performs a product of its inputs.

The links between the rule layer and the layers before and after it have unit weight.

3) The Output Membership Layer: Each node in this layer takes inputs from all rules that conclude this output membership function and outputs the sum of the weights for that output fuzzy set.

4) The Defuzzification Layer: This layer performs a defuzzification by normalizing the weights from each output membership function and performing a convex combination with the peaks of the output membership functions. This implements the singleton method previously defined.

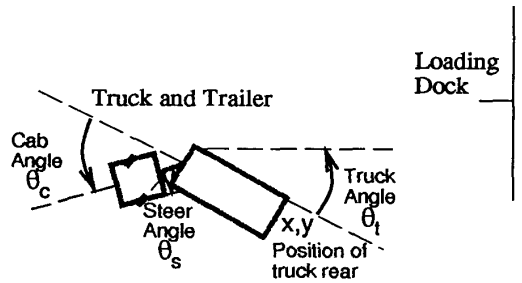
5) Lateral Inhibitory Connections: These connections are used to solve a problem with the standard fuzzy inference techniques when used with dependent rules. Consider the example rule set below, as represented in network form in Fig. 6:

1) output = low
2) IF input1 = high THEN output = high
3) IF input1 = high AND input2 = low THEN output = med.

Each of the rules if correct *independently*. It is only in combination that they conflict. If we use standard fuzzy techniques to compute the output, rule 1 will add its contribution to rules 2 and 3 to drive the output lower than it should be, even though we know that along the input1 = high axis, the output should be high. Similarly, rule 2 will pull the output higher than it should be at the input1 = high and input2 = low corner. We know specifically what the value at this corner should be, and the interference of the other rules is unsatisfactory.

What the ideal inference technique would do is the following: in the corner input1 = high and input2 = low, we know from rule 3 that the output should be medium. We are not interested in the contribution of the other two rules. Similarly, if we are along the input1 = high axis (but not too near input2 = low), we wish the output to be high because of rule 2. If we are not too near the input1 = high axis, only rule 1 applies and the output should be low. We also wish a smooth tradeoff between these regions, in keeping with the basic principles of fuzzy logic.

What we really want is that a more general rule dependent on a more specific rule should only be allowed to fire *to the degree that the more specific rule is not firing*. Thus, the degree of firing of rule 3 should gate the maximum firing allowed for rule 2. Both rules should



have a similar effect on rule 1. Let the degree of firing of rule i be called f_i, and the input to the output membership functions layer be called o_i. Then we can express this relationship as

$$o_1 = f_1(1 - f_3)(1 - f_2)$$
$$o_2 = f_2(1 - f_3)$$
$$o_3 = f_3.$$

Thus, at the corner specified by rule 3, it alone is allowed to fire, while rules 1 and 2 are completely shut off. This is expressed in network form in the links between the rule layer and the output membership functions layer. The lateral arrows are inhibitory connections which take the value at their input, invert it (subtract it from one), and multiply it by the value at their output. More generally, each rule has a lateral inhibitory link coming to it from every higher-order rule which contains all of its conditions.

IV. Experimental Results

In this section, we demonstrate our function approximation system by converting a hand-crafted neural controller for the truck backer-upper problem (Fig. 7) into a fuzzy one.

Jenkins and Yuhas [13] have developed by hand a very efficient neural network for solving the problem of backing up a truck and trailer to a loading dock (Fig. 8). Its trajectory is comparable to that of other truck backer-upper systems. We have chosen this system as a function to approximate because it is highly nonlinear and difficult to represent in terms of fuzzy rules. The equations of motion used for the truck are shown below:

$$\dot{x} = B\cos(\theta_s)\cos(\theta_c)\cos(\theta_t)$$
$$\dot{y} = B\cos(\theta_s)\cos(\theta_c)\sin(\theta_t)$$
$$\dot{\theta}_t = -B/L_t\cos(\theta_s)\sin(\theta_c)$$
$$\dot{\theta}_c = -\dot{\theta}_t + B/L_c\sin(\theta_s)$$

where

- $B = 0.2$ m/timestep is the backing velocity of the trailer
- $L_t = 14.0$ m is the length of the trailer
- $L_c = 6.0$ m is the length of the cab and tongue
- x and y are the coordinates of the center of the back of the trailer
- θ_c is the cab angle
- θ_t is the truck angle
- θ_s is the steering angle (output of the controller).

The truck moves in a field of size 80 m by 80 m.

The function approximation system was trained on 245 example runs of the Jenkins–Yuhas controller, with initial states distributed symmetrically about the goal state. At each simulation timestep, the truck state variables and the resulting control output were recorded. The concatenation of these data from all 245 runs was the input to the function approximator. In order to show the effect of varying the number of membership functions, we have fixed the maximum

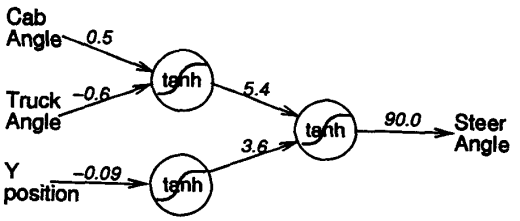


Fig. 8. The Jenkins–Yuhas network.

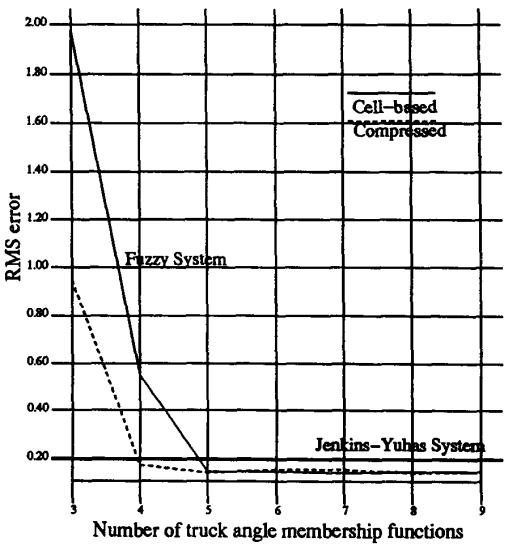


Fig. 9. Error in final y position of truck backer-upper.

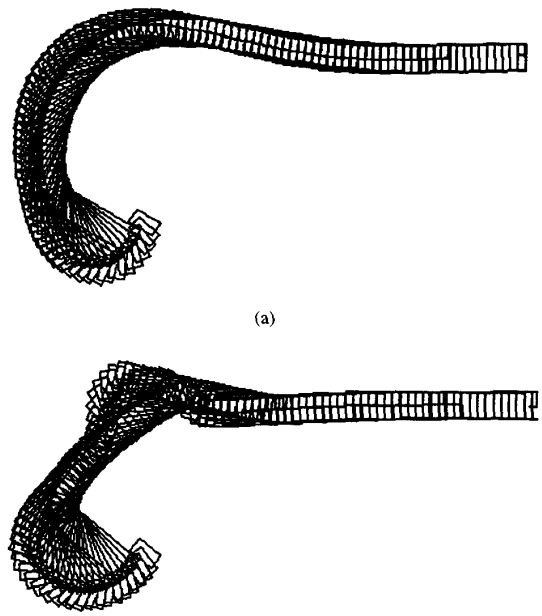
TABLE I
NUMBER OF RULES AND COMPRESSION FOR LEARNED TBU SYSTEMS

	Number of truck angle membership functions						
	3	4	5	6	7	8	9
# Cell-Based	75	100	125	150	175	200	225
# Compressed	48	67	86	100	114	138	154
Compression	36%	33%	31%	33%	35%	31%	32%

number of membership functions for the y-position and cab angle at 5 and set the maximum allowable error to zero, thus guaranteeing that the system will fill out all of the allowed membership functions. We varied the maximum number of truck angle membership functions from 3 to 9. The effects of this are shown in Fig. 9. Note that the error decreases sharply and then holds constant, reaching its minimum at 5 membership functions. The Jenkins–Yuhas network performance is shown as a horizontal line. At its best, the fuzzy system performs slightly better than the system it is approximating.

For this experiment, we set a goal of 33% rule compression. We manually varied the parameter α in the L-measure for each rule set to get the desired compression. Note (in Fig. 9) the performance of the system with compressed rules. The performance is in every case almost identical to that of the original cell-based rule sets. This validates the effectiveness of our rule compression and dependent rule inference schemes. The number of rules and the amount of rule compression obtained can be seen in Table I.

One thing we have not quantified in this example is the *smoothness* of the truck trajectory (see Fig. 10). While the learned fuzzy system with 5 truck angle membership functions actually performs better in rms docking error than the original Jenkins–Yuhas network, its path is not as smooth. The fuzzy truck backer-upper has "modes" of operation: the truck will first turn around, then back up in a straight line at a diagonal angle, then change direction sharply and back towards the loading dock. This is directly related to the piecewise approximation to the original function. This piecewise approximation is also incidentally responsible for the slightly improved performance—while the Jenkins–Yuhas network approaches the loading dock *asymptotically*, the fuzzy system turns sharply to line up with it.



(a)

(b)

Fig. 10. Demonstration of mode-based behavior of fuzzy system. (a) Jenkins–Yuhas hand-crafted neural system. (b) Learned fuzzy system.

V. SUMMARY AND DISCUSSION

We have presented a method which, given examples of a function and its independent variables, can construct a computational network based on fuzzy logic to predict the function given the independent variables. The user must only specify the maximum number of membership functions for each variable and/or the maximum rms error from the example data.

There are three innovative aspects of this system, each of which is valuable independently:

- *Membership functions are generated automatically.* Membership functions are most often generated by hand. This scheme allows the membership functions to be chosen based only upon an error criterion by an algorithm which must terminate in a small number of steps.
- *Cell-based rules are compressed into a minimal rule set.* Many systems exist using cell-based rule sets. The ability to compress such rule sets and retain the same performance will lead to more manageable, understandable rule sets.
- *The problem of inference with dependent rules is solved.* When a system designer sets up a fuzzy system, he or she may well want to use dependent rules. The proposed inference scheme allows the rule system to perform as expected.

We have applied our function approximation system to the conversion of any existing controller into a fuzzy controller. This application begs the question "What use is this when there exists no working controller?" In our ongoing research, we are using reinforcement learning to adapt a table-based controller to a performance criterion. We then extract a fuzzy controller from the learned table-based controller using the techniques described in this paper, for simplicity of representation and explanation ability. This combination of reinforcement learning and fuzzy function approximation will allow a fuzzy controller to be synthesized for a completely unknown plant. We expect to report on this in the very near future.

REFERENCES

[1] D. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, D. Rumelhart and J. McClelland, Eds. Cambridge, MA: MIT Press, 1986.

[2] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proc. IEEE*, vol. 78, Sept. 1990.

[3] B. Kosko, *Neural Networks and Fuzzy Systems*. Englewood Cliffs, NJ: Prentice Hall, 1992.

[4] C. Lin and C. Lee, "Neural-network-based fuzzy logic control and decision system," *IEEE Trans. Comput.*, vol. 40, pp. 1320–36, Dec. 1991.

[5] S. Horikawa, T. Furuhashi, and Y. Uchikawa, "On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm," *IEEE Trans. Neural Networks*, vol. 3, pp. 801–806, Sept. 1992.

[6] F. d'Alché Buc, V. Andrès, and J. Nadal, "Learning fuzzy control rules with a fuzzy neural network," *Proc. Int. Conf. Artificial Neural Networks*, 1992.

[7] L. Wang and J. Mendel, "Fuzzy basis functions, universal approximation, and orthogonal least-squares learning," *IEEE Trans. Neural Networks*, vol. 3, Sept. 1992.

[8] M. Sugeno and G. Kang, "Structure identification of fuzzy model," *Fuzzy Sets Syst.*, vol. 28, pp. 15–33, 1988.

[9] M. Sugeno, "Fuzzy control: Principles, practice and perspectives," presented at the *IEEE Int. Conf. Fuzzy Syst.*,Mar. 1992.

[10] C. Higgins and R. Goodman, "Incremental learning using rule-based neural networks," *Proc. Int. Joint Conf. Neural Networks*, vol. 1, pp. 875–880, July 1991.

[11] R. Goodman and P. Smyth, "Information-theoretic rule induction," in *Proc. Euro. Conf. Artificial Intell.*, Munich, Germany.

[12] R. Goodman, C. Higgins, J. Miller, and P. Smyth, "Rule-based networks for classification and probability estimation," *Neural Computat.*, vol. 4, Nov. 1992.

[13] R. Jenkins and B. Yuhas, "A simplified neural-network solution through problem decomposition: The case of the truck backer-upper," *Neural Computat.*, vol. 4, Sept. 1992.